Linear and Affine Transformations
Coordinate Systems

Recall

A transformation T is **linear** if

- $T(v + w) = T(v) + T(w)$
- $T(\alpha v) = \alpha T(v)$

Recall

A transformation T is **linear** if

- $T(v + w) = T(v) + T(w)$
- $T(\alpha v) = \alpha T(v)$

Every linear transformation can be represented as matrix

Linear Transformation Examples

Uniform Scaling

Non-uniform Scaling

Rotations

Reflections

Orthogonal Projections

…

Translations?

Problem with Translation

Translation by (t_x, t_y, t_z) not linear!

$$T(\alpha v) = (\alpha v_x + t_x, \alpha v_y + t_y, \alpha v_z + t_z)$$

$$\alpha T(v) = (\alpha v_x + \alpha t_x, \alpha v_y + \alpha t_y, \alpha v_z + \alpha t_z)$$

Would like a unified framework for handling all transformations…

Homogeneous Coordinates

Main idea: add a dummy 4th dimension

- points: $(x, y, z) \to (x, y, z, 1)$
- vectors: $(x, y, z) \to (x, y, z, 0)$

In Homogeneous Coordinates

$$\begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} p_x \\ p_y \\ p_z \\ 1 \end{bmatrix} =$$

In Homogeneous Coordinates

$$\begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} p_x \\ p_y \\ p_z \\ 1 \end{bmatrix} = \begin{bmatrix} p_x + t_x \\ p_y \\ p_z \\ 1 \end{bmatrix}$$

In Homogeneous Coordinates

$$\begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} p_x \\ p_y \\ p_z \\ 1 \end{bmatrix} = \begin{bmatrix} p_x + t_x \\ p_y \\ p_z \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} v_x \\ v_y \\ v_z \\ 0 \end{bmatrix} =$$

In Homogeneous Coordinates

$$\begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} p_x \\ p_y \\ p_z \\ 1 \end{bmatrix} = \begin{bmatrix} p_x + t_x \\ p_y \\ p_z \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} v_x \\ v_y \\ v_z \\ 0 \end{bmatrix} = \begin{bmatrix} v_x \\ v_y \\ v_z \\ 0 \end{bmatrix}$$

Homogeneous Coordinates

Main idea: add a dummy 4^{th} dimension

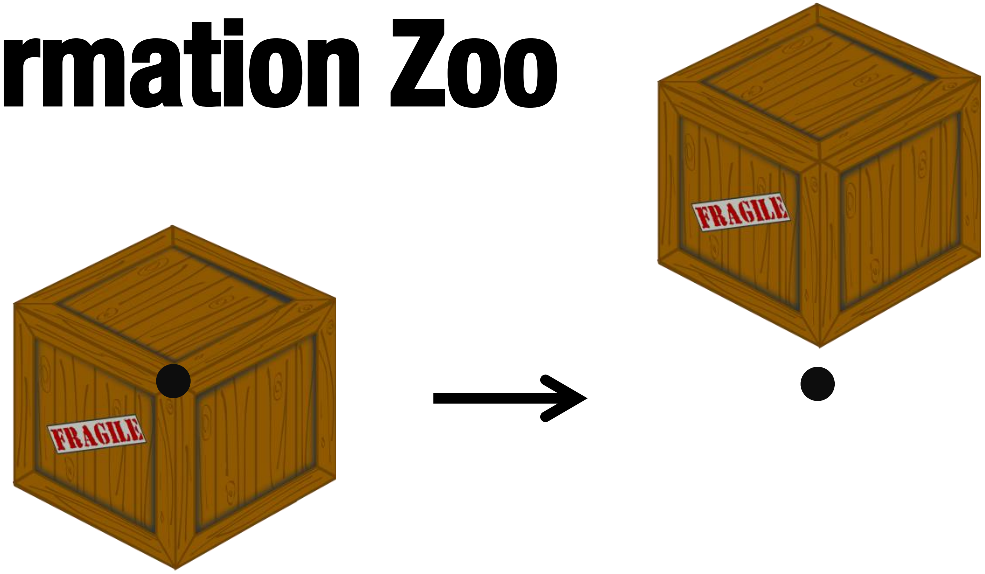
- points: $(x, y, z) \to (x, y, z, 1)$
- vectors: $(x, y, z) \to (x, y, z, 0)$

Now translation **is** matrix multiplication!

4 x 4 matrix transformations called **affine**

Linear Transformation Zoo

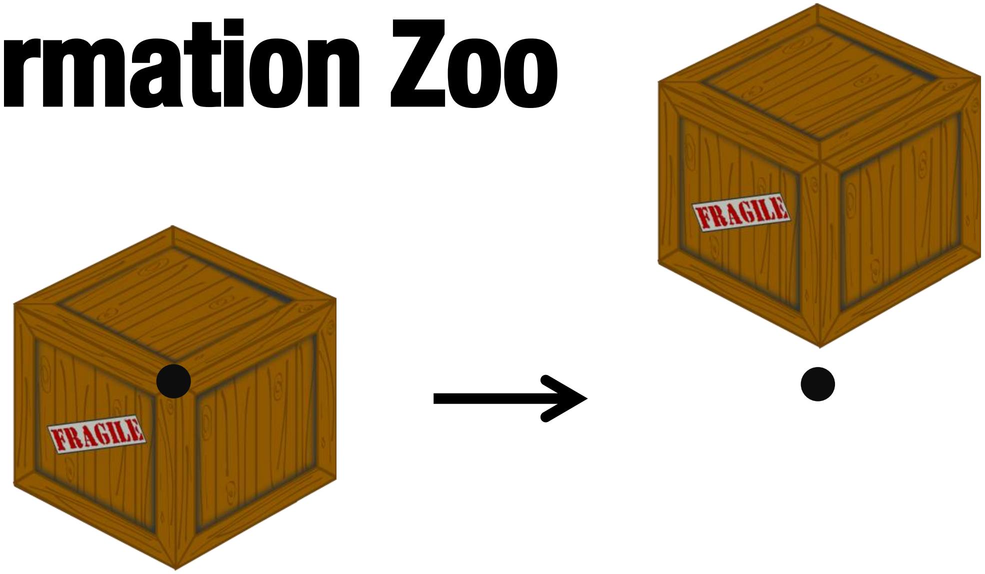
Translation:



$$T = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Linear Transformation Zoo

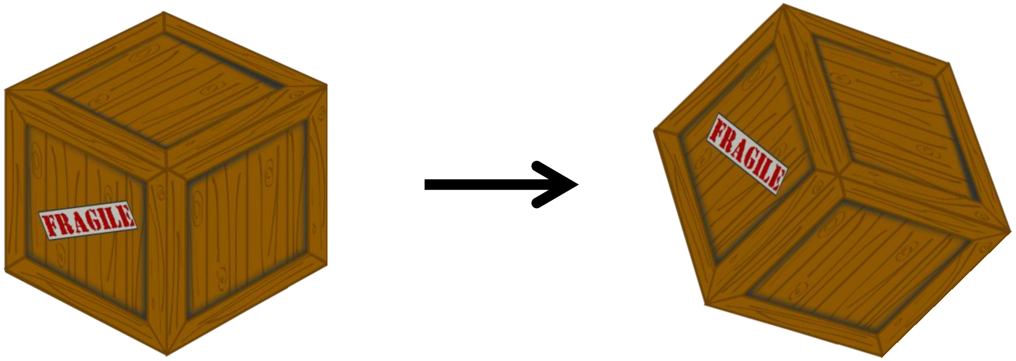
Translation:



$$T = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad T^{-1} = \begin{bmatrix} 1 & 0 & 0 & -t_x \\ 0 & 1 & 0 & -t_y \\ 0 & 0 & 1 & -t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Linear Transformation Zoo

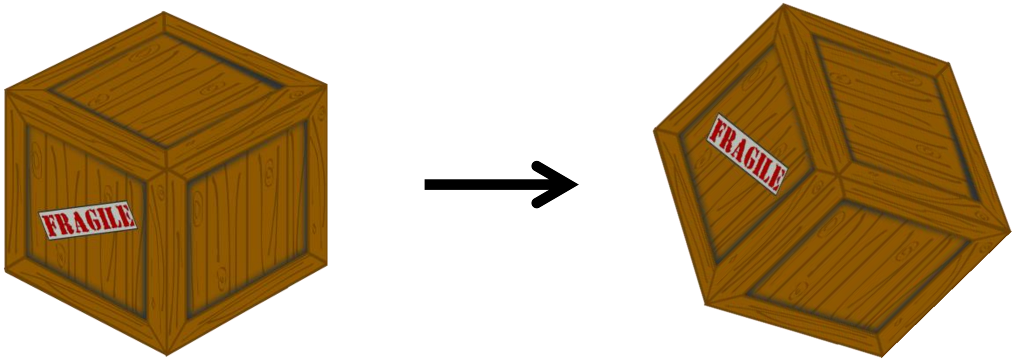
Rotation:



$$R = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix}$$

Linear Transformation Zoo

Rotation:

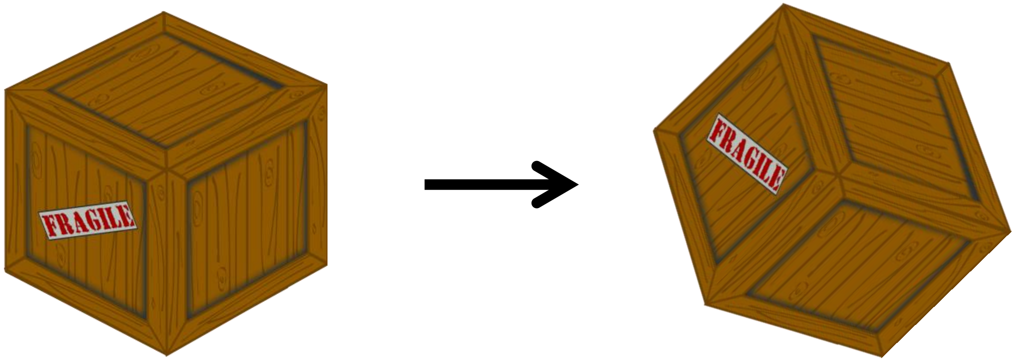


$$R = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix}$$

$$R^T R = I$$

Linear Transformation Zoo

Rotation:



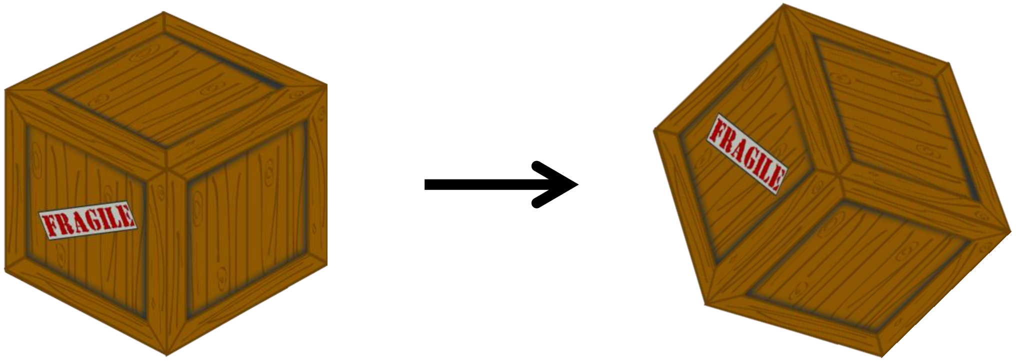
$$R = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix}$$

what about in homogeneous coordinates?

$$R^T R = I$$

Linear Transformation Zoo

Rotation:

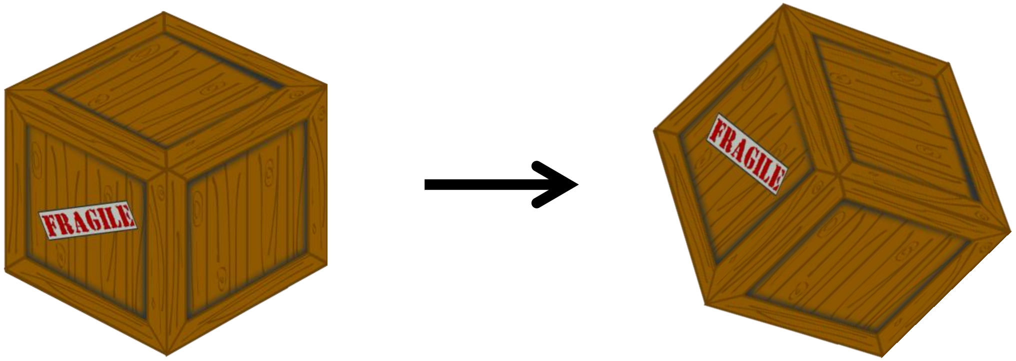


$$R = \begin{bmatrix} r_{11} & r_{12} & r_{13} & 0 \\ r_{21} & r_{22} & r_{23} & 0 \\ r_{31} & r_{32} & r_{33} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$R^T R = I$$

Linear Transformation Zoo

Rotation:

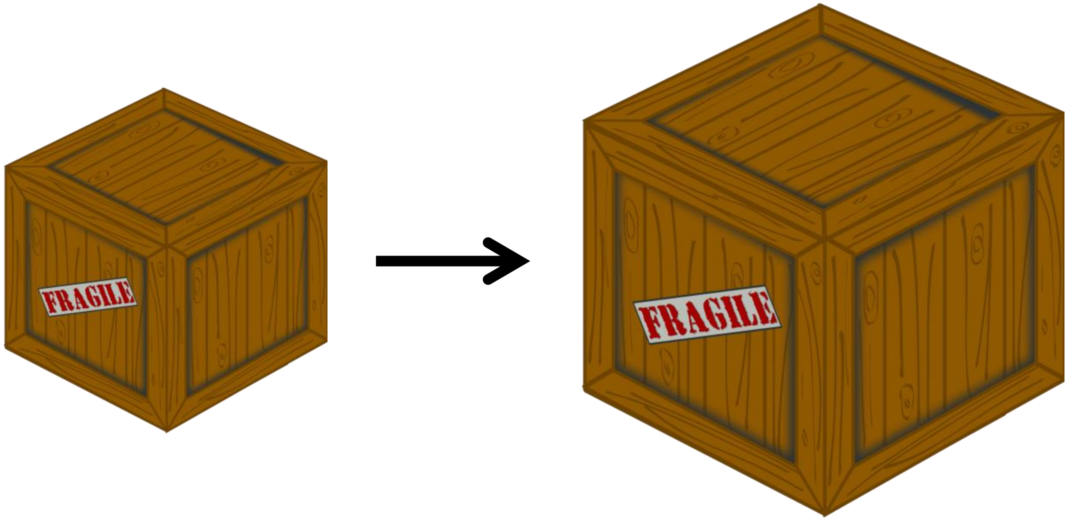


$$R = \begin{bmatrix} r_{11} & r_{12} & r_{13} & 0 \\ r_{21} & r_{22} & r_{23} & 0 \\ r_{31} & r_{32} & r_{33} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad R^{-1} = R^T$$

$$R^T R = I$$

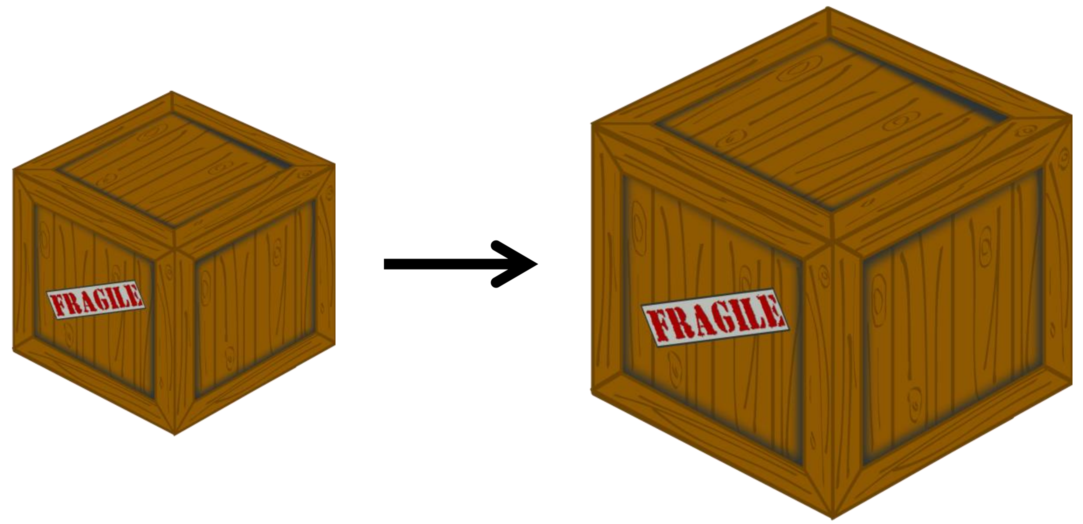
Linear Transformation Zoo

Uniform scaling:



Linear Transformation Zoo

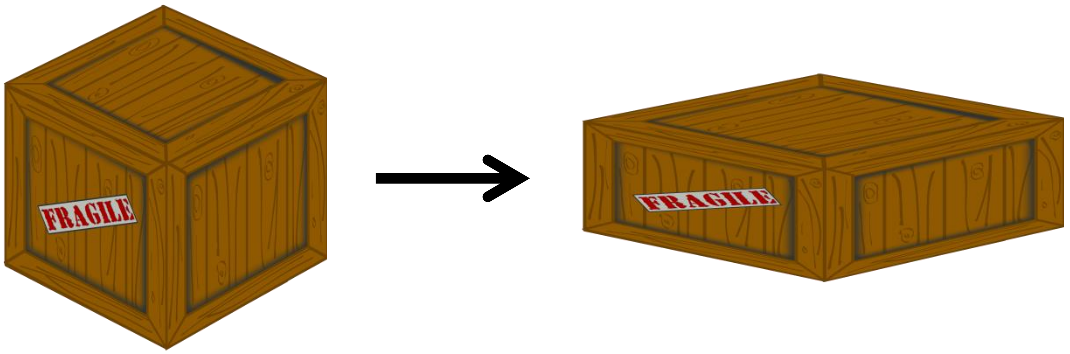
Uniform scaling:



$$S = \begin{bmatrix} s & 0 & 0 & 0 \\ 0 & s & 0 & 0 \\ 0 & 0 & s & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad S^{-1} = \begin{bmatrix} \frac{1}{s} & 0 & 0 & 0 \\ 0 & \frac{1}{s} & 0 & 0 \\ 0 & 0 & \frac{1}{s} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Linear Transformation Zoo

Scaling:



$$S = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_y & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} S^{-1} = \begin{bmatrix} \frac{1}{s_x} & 0 & 0 & 0 \\ 0 & \frac{1}{s_y} & 0 & 0 \\ 0 & 0 & \frac{1}{s_z} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

What About Non-Axis-Aligned?

What About Non-Axis-Aligned?

compose transformations!

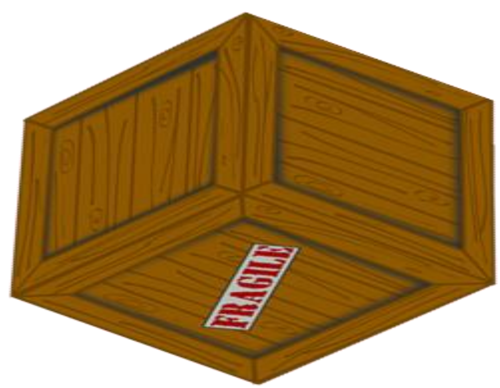
What About Non-Axis-Aligned?

compose transformations!

I

R

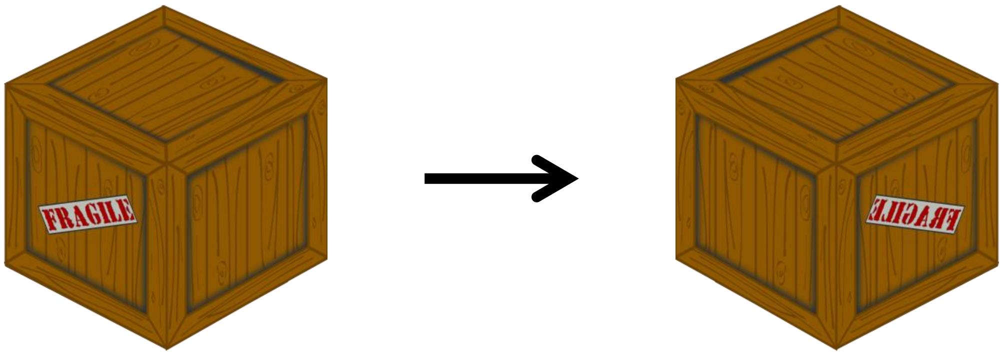
SR



$R^T S R$

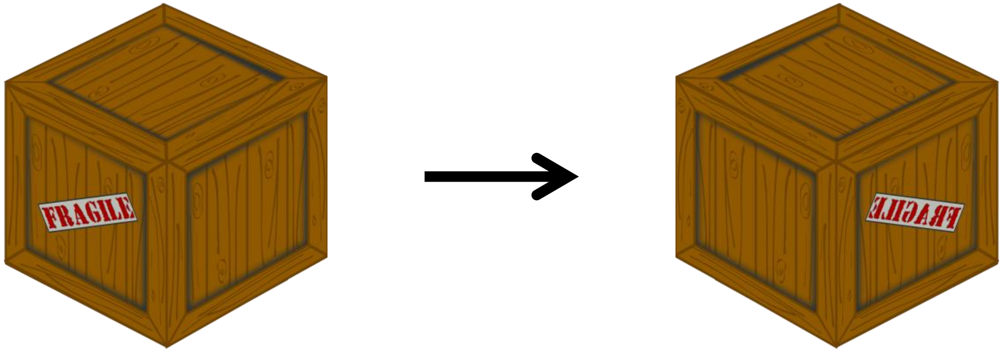
Linear Transformation Zoo

Reflection:



Linear Transformation Zoo

Reflection:

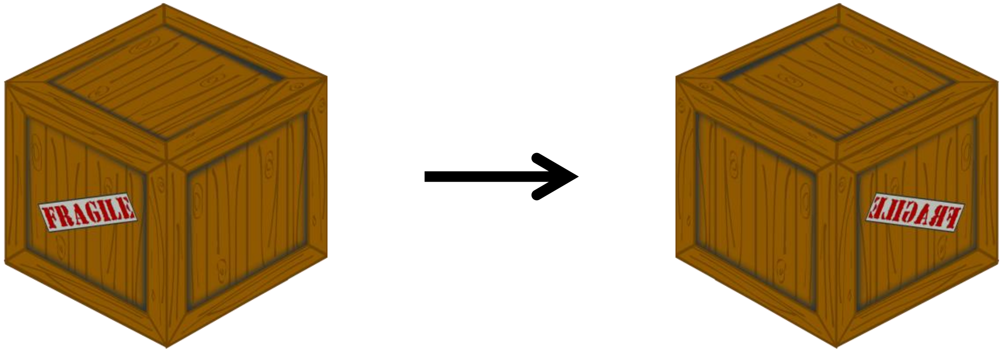


$$Rf = \begin{bmatrix} -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

axis to reflect

Linear Transformation Zoo

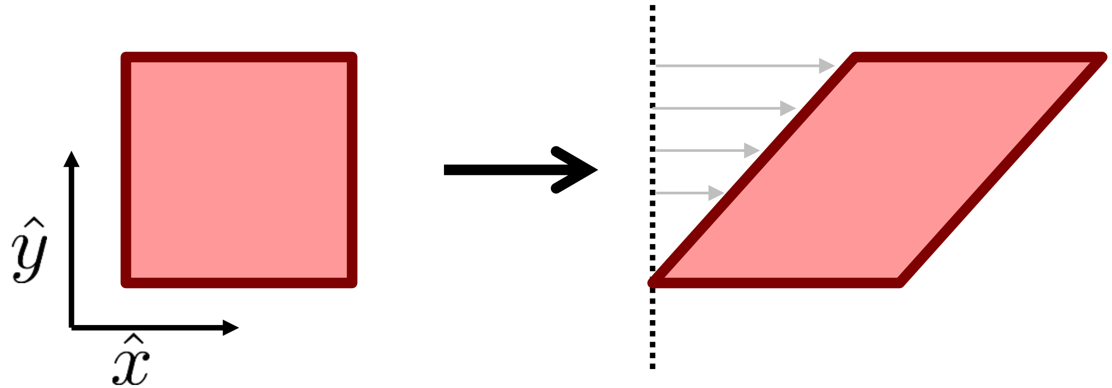
Reflection:



$$Rf = \begin{bmatrix} -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad Rf^{-1} = \begin{bmatrix} -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

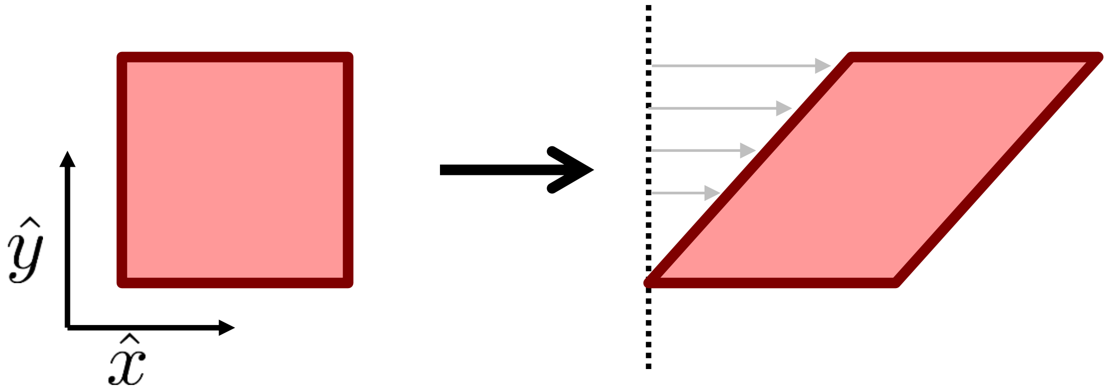
Linear Transformation Zoo

Shear:



Linear Transformation Zoo

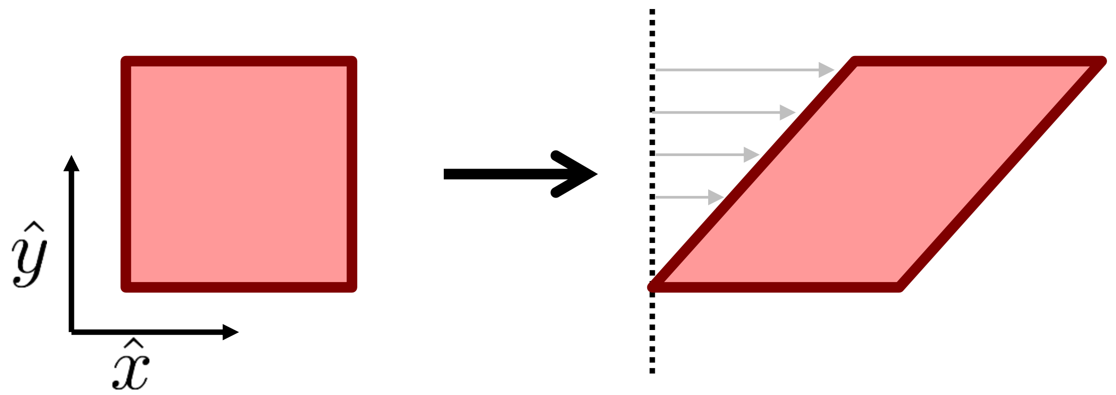
Shear:



$$Sh = \begin{bmatrix} 1 & sh \\ 0 & 1 \end{bmatrix}$$

Linear Transformation Zoo

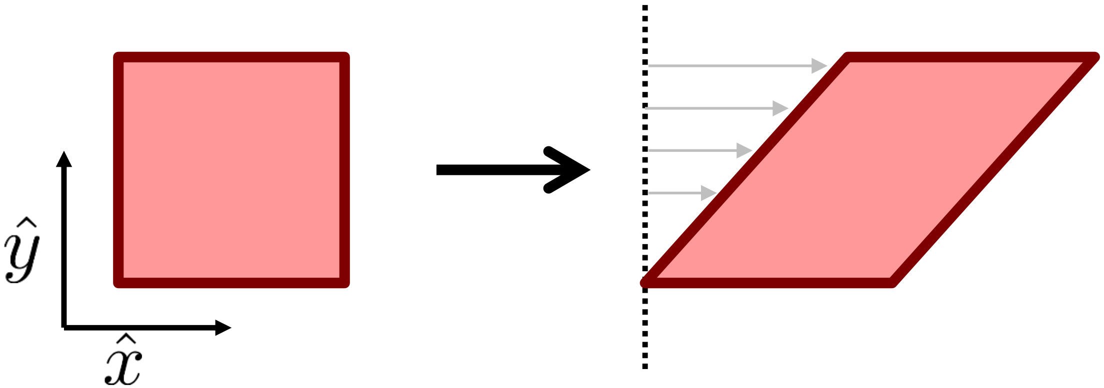
Shear:



$$\begin{array}{rcl} & \text{shear y-axis (column 2)} & \\ Sh = & \begin{bmatrix} 1 & sh & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} & \begin{array}{l} \leftarrow \text{in x-axis direction} \\ \\ \\ \\ \end{array} \end{array}$$

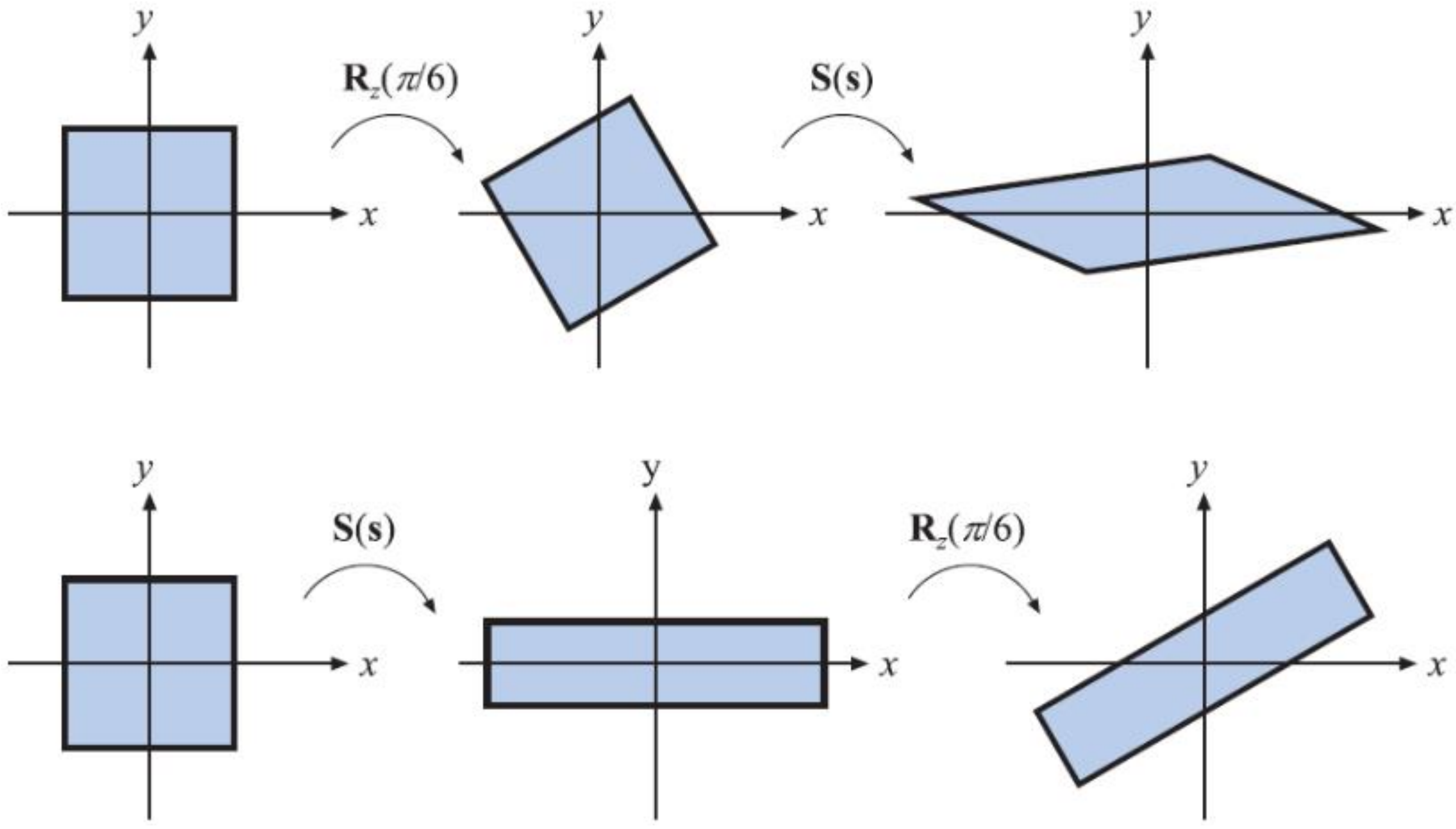
Linear Transformation Zoo

Shear:



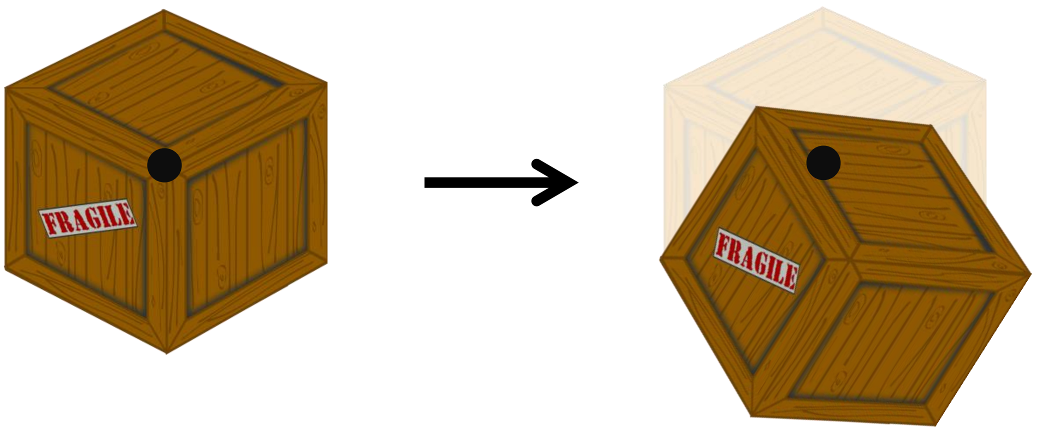
$$Sh = \begin{bmatrix} 1 & sh & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad Sh^{-1} = \begin{bmatrix} 1 & -sh & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Combining Transformations

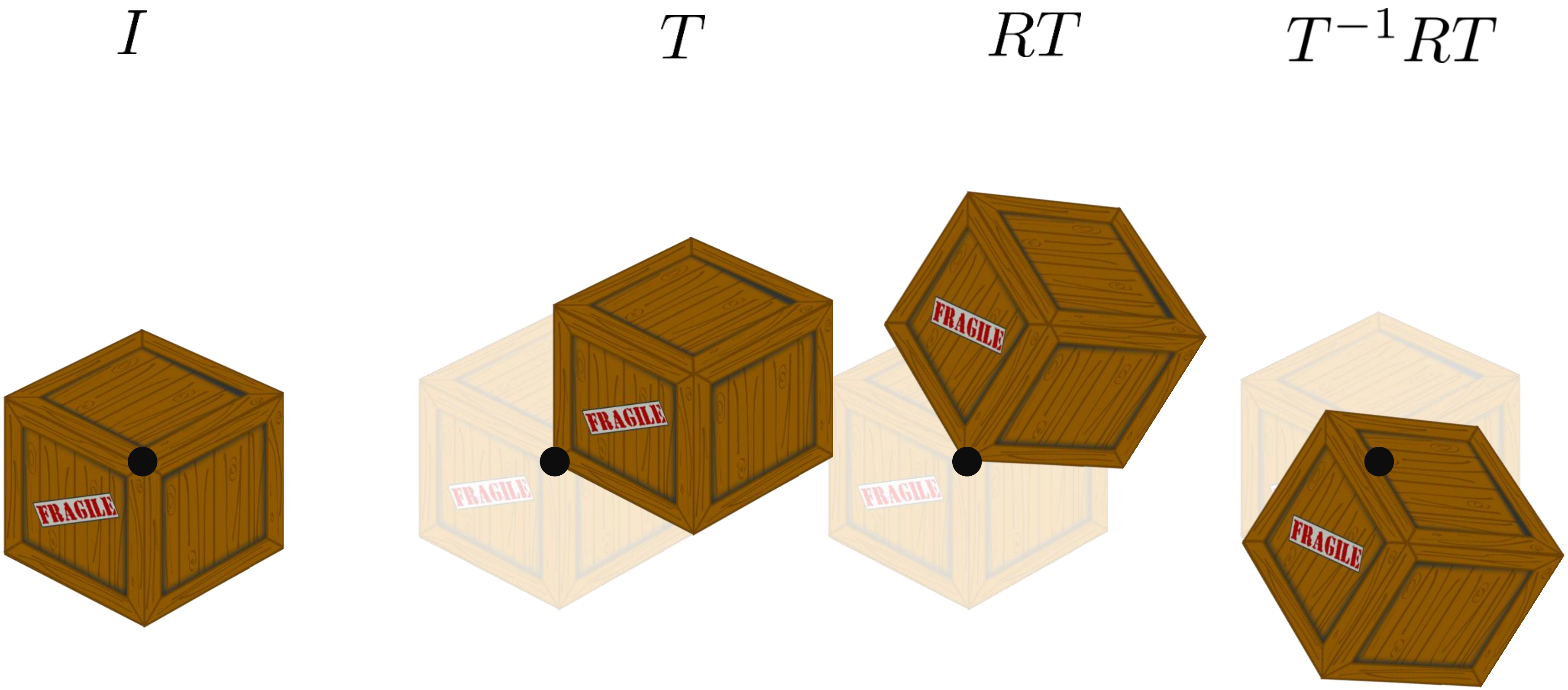


matrix multiplication **does not commute**

Example: Rotate About Point

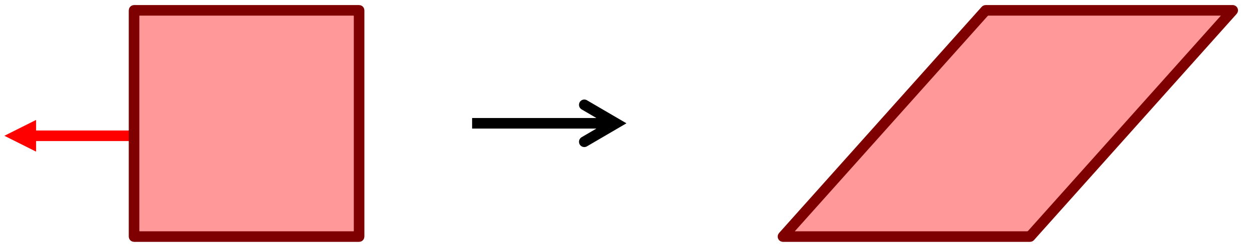


Example: Rotate About Point



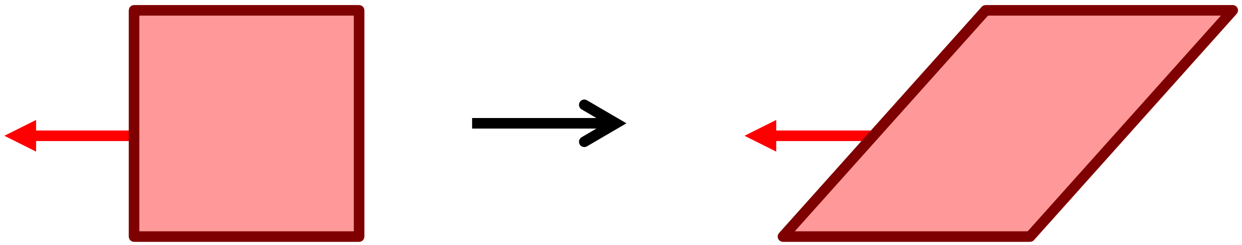
Transforming Normals

The problem:



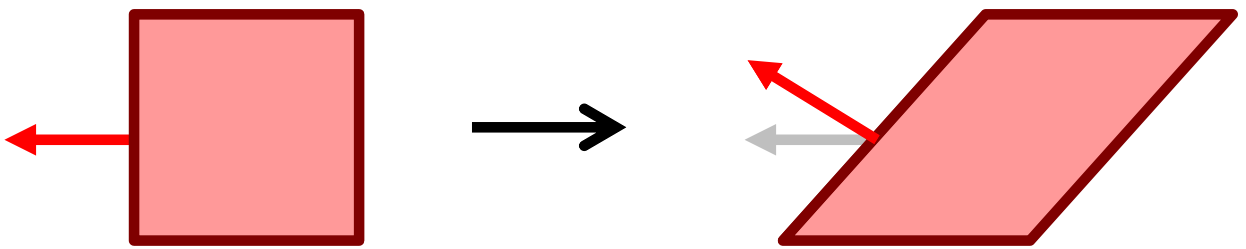
Transforming Normals

The problem:



Transforming Normals

The problem:

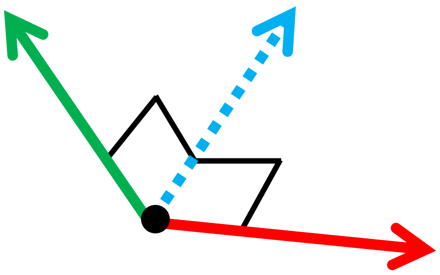


Points and vectors: T

Normals: $T^{-T} = \left(T^{-1}\right)^{T}$

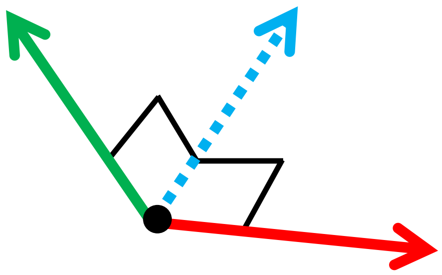
What is a Coordinate System?

1. an **origin**
2. a **frame** of vectors spanning space



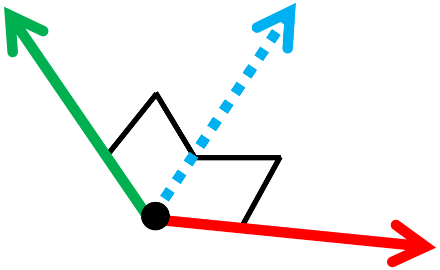
What is a Coordinate System?

1. an **origin**
2. a **frame** of vectors spanning space
    - usually orthonormal
    - usually right-handed



What is a Coordinate System?

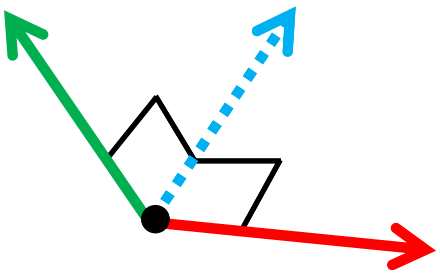
1. an **origin**
2. a **frame** of vectors spanning space
    - usually orthonormal
    - usually right-handed



How represented?

What is a Coordinate System?

1. an **origin**
2. a **frame** of vectors spanning space
   - usually orthonormal
   - usually right-handed



How represented?

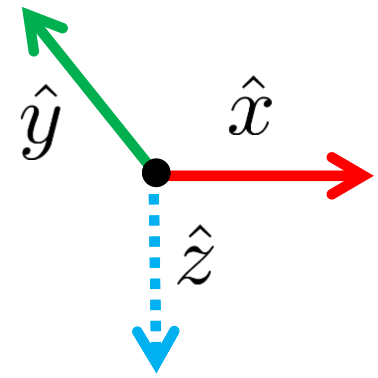
- in other coordinates…

(turtles all the way down?)

Cartesian "World" Coordinates

Canonical "root" coordinate system

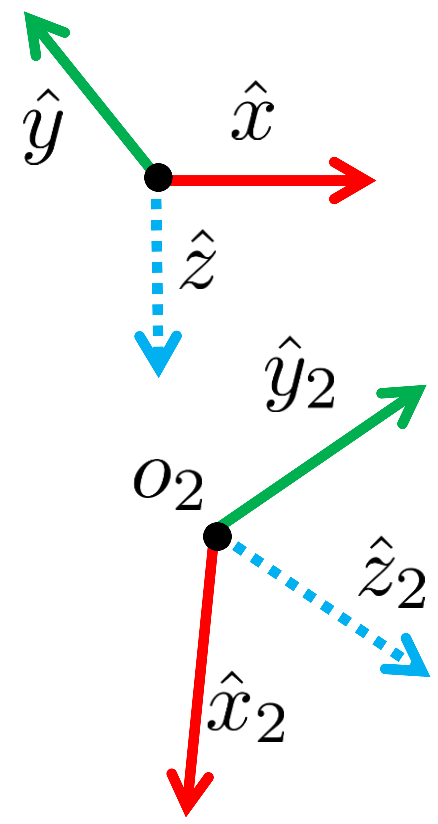
Usually y points "up," x and z "horizontal"



But this is arbitrary

Transforming Coordinate Systems

Can define coordinate system **in terms of** world coordinates

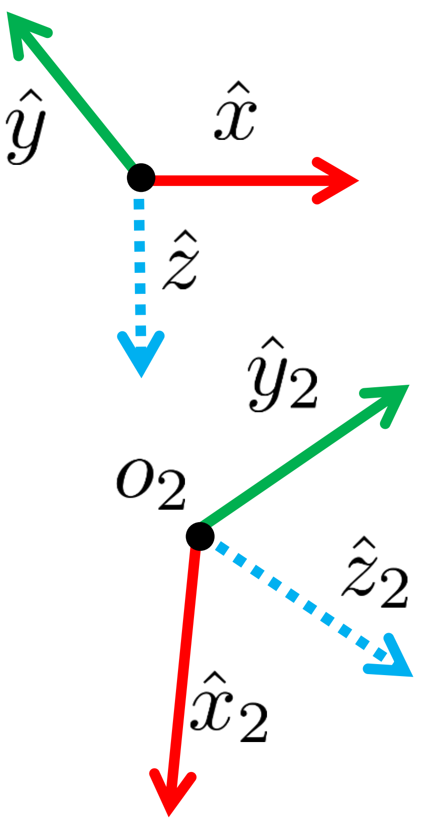


Transforming Coordinate Systems

Can define coordinate system **in terms of** world coordinates

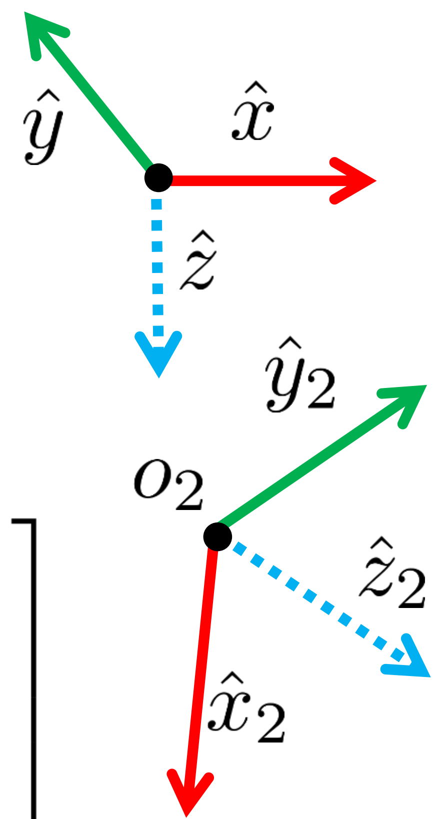
Given $o_2, \hat{x}_2, \hat{y}_2, \hat{z}_2$ in world coords

$$(a, b, c)_{\text{world}} = o_2 + a\hat{x}_2 + b\hat{y}_2 + c\hat{z}_2$$



Transforming Coordinate Systems

Can define coordinate system **in terms of** world coordinates



Given $o_2, \hat{x}_2, \hat{y}_2, \hat{z}_2$ in world coords

$$(a, b, c)_{\text{world}} = o_2 + a\hat{x}_2 + b\hat{y}_2 + c\hat{z}_2$$

$$(a, b, c)_{\text{world}} = \left[\begin{array}{ccc|c} \hat{x}_2 & \hat{y}_2 & \hat{z}_2 & o_2 \\ \hline 0 & 0 & 0 & 1 \end{array}\right] \begin{bmatrix} a \\ b \\ c \\ 1 \end{bmatrix}$$

Change of Coordinates Matrix

$$(a, b, c)_{\text{world}} = \boxed{\left[\begin{array}{ccc|c} \hat{x}_2 & \hat{y}_2 & \hat{z}_2 & o_2 \\ \hline 0 & 0 & 0 & 1 \end{array}\right]} \begin{bmatrix} a \\ b \\ c \\ 1 \end{bmatrix}$$

Maps **from** local **to** world coordinates

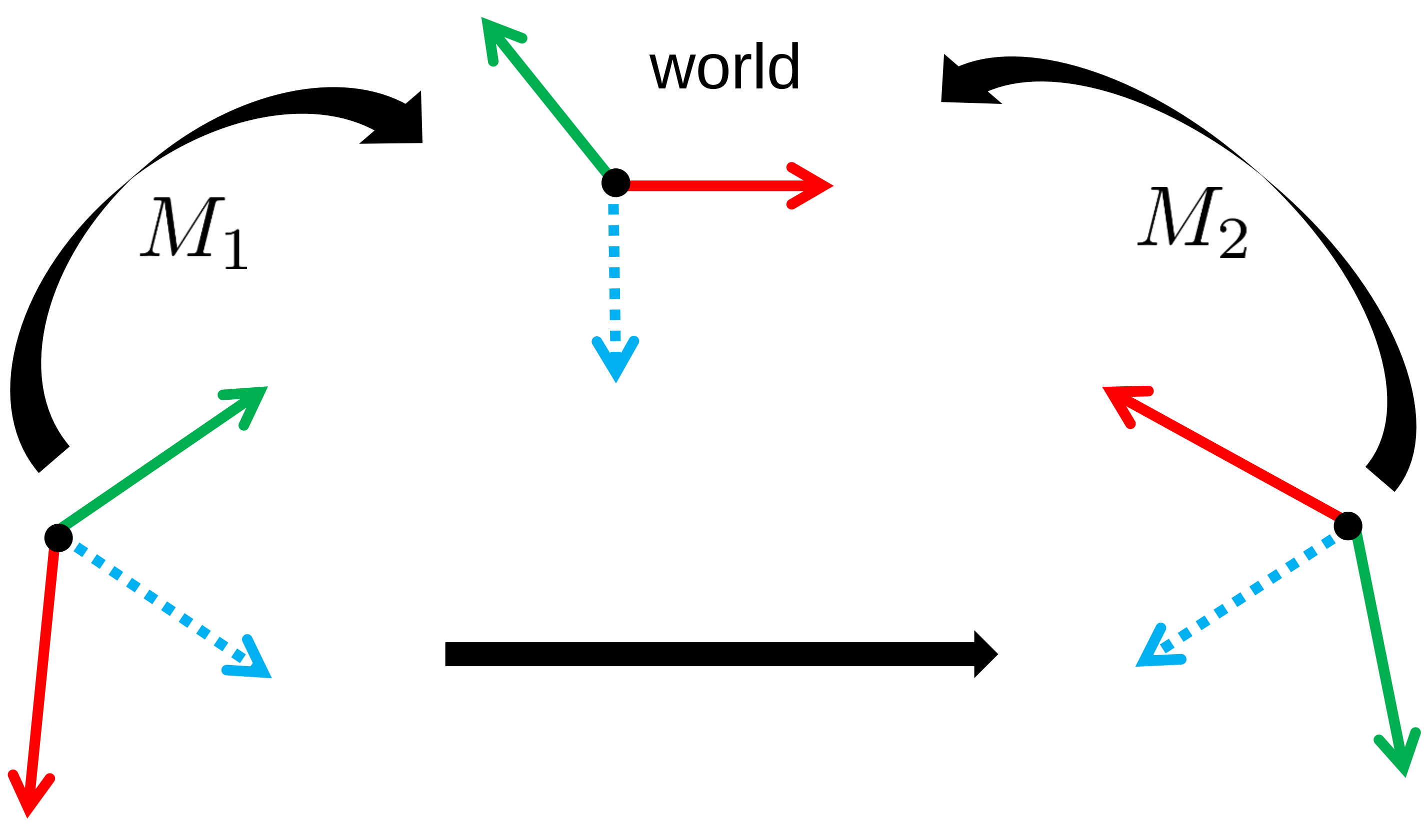
Change of Coordinates Matrix

$$(a,b,c)_{\text{world}} = \boxed{\left[\begin{array}{ccc|c} \hat{x}_2 & \hat{y}_2 & \hat{z}_2 & o_2 \\ \hline 0 & 0 & 0 & 1 \end{array}\right]} \begin{bmatrix} a \\ b \\ c \\ 1 \end{bmatrix}$$

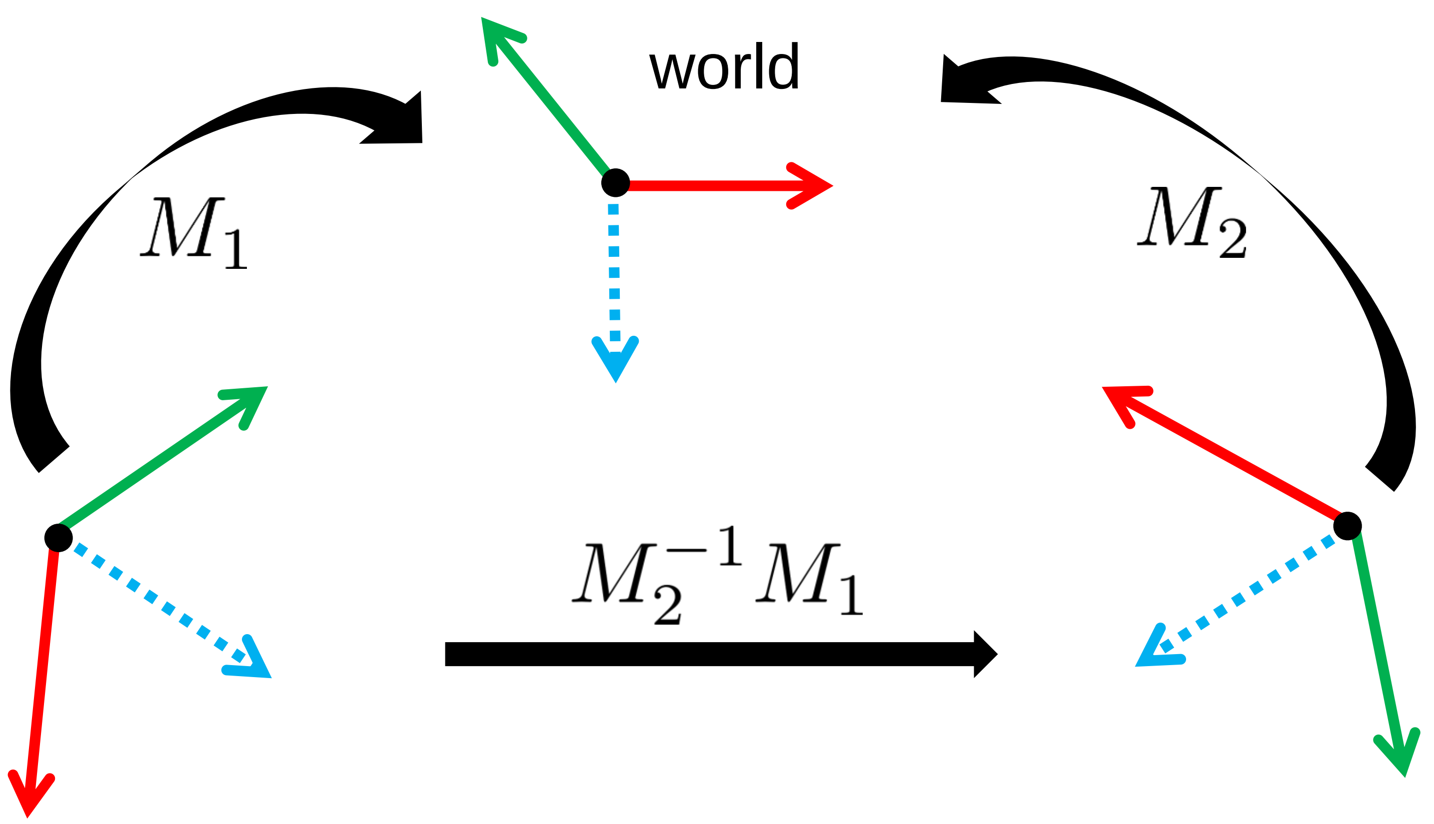
Maps **from** local **to** world coordinates

How to map back?

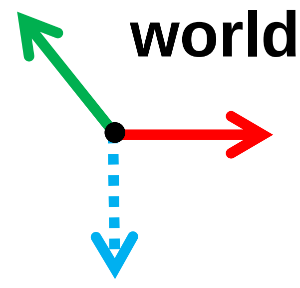
More Coordinates Systems

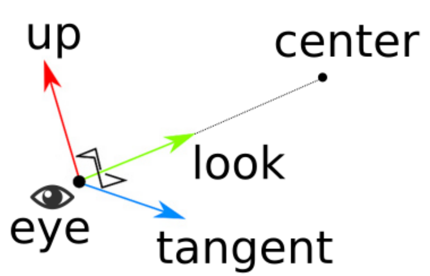


More Coordinates Systems



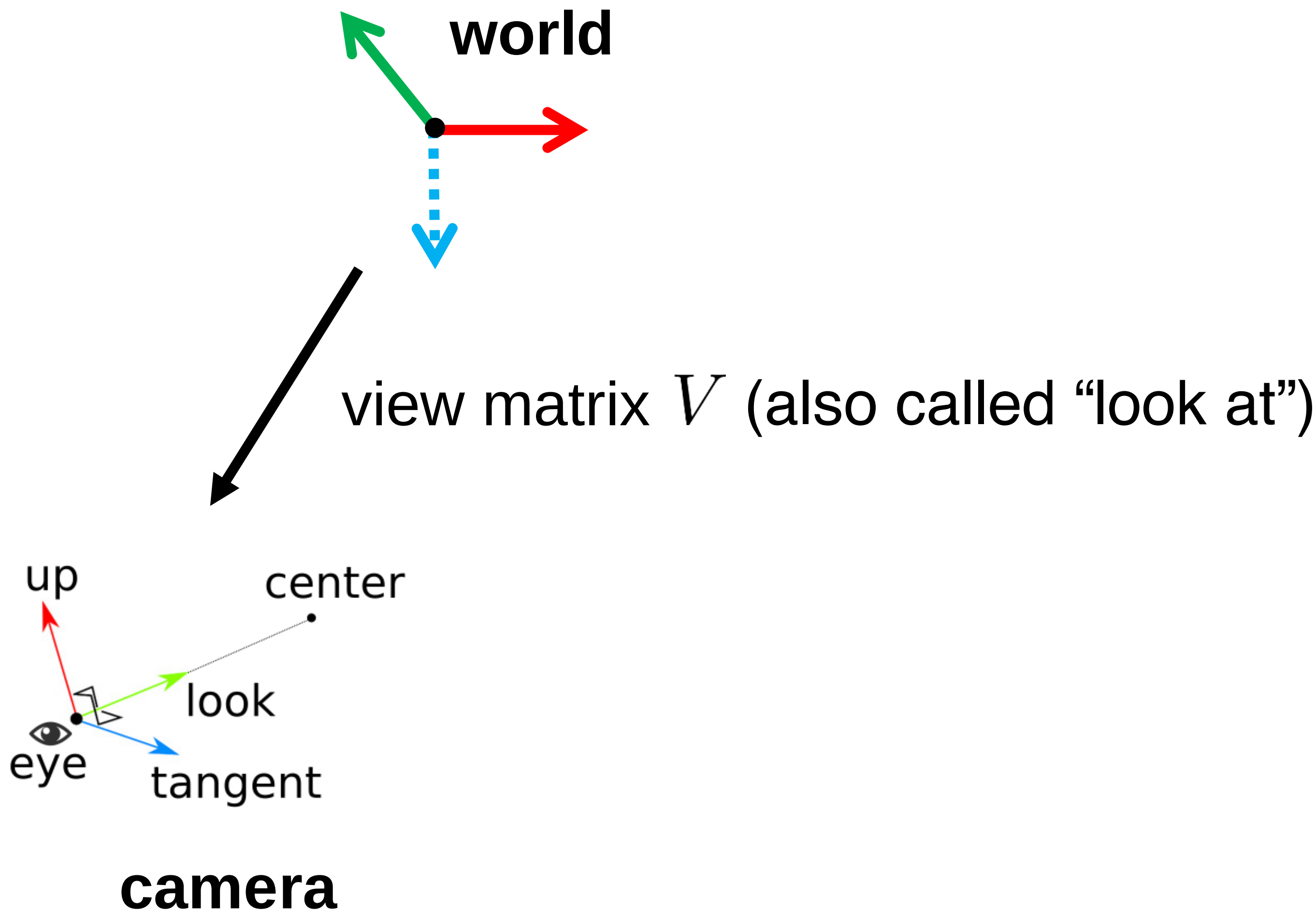
Coordinate Systems in Graphics





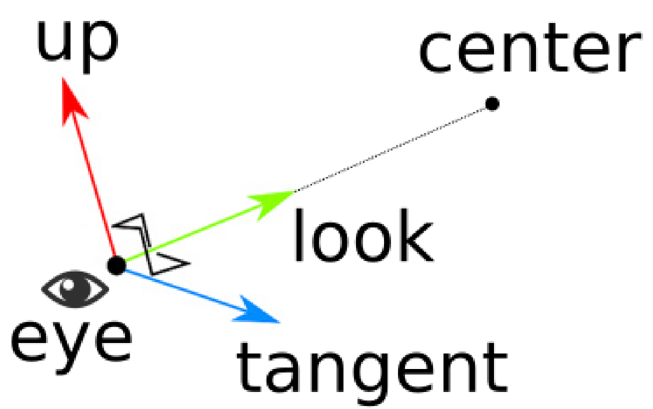
camera

Coordinate Systems in Graphics



Building the View Matrix

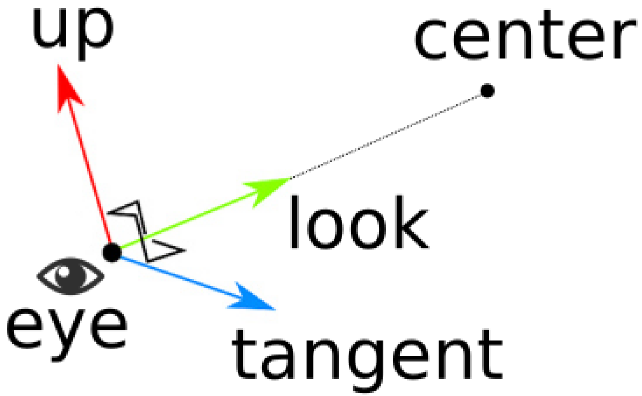
Three axes: tangent, up, look



Building the View Matrix

Three axes: tangent, up, look

Note: camera looks down **negative** look direction for extra confusion

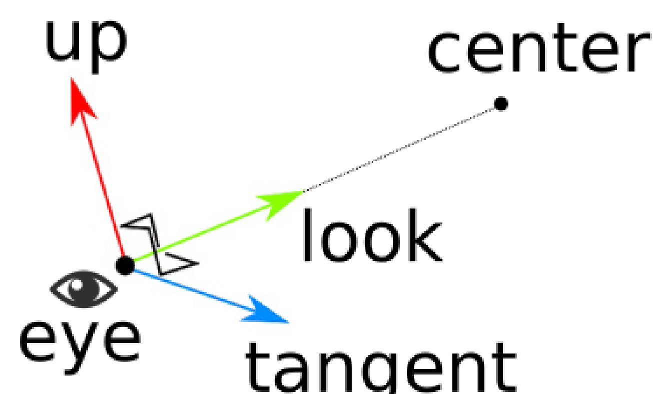


Building the View Matrix

Three axes: tangent, up, look

Note: camera looks down **negative** look direction for extra confusion

$$V = \left[\begin{array}{ccc|c} \text{tangent} & \text{up} & \text{-look} & \text{eye} \\ \hline 0 & 0 & 0 & 1 \end{array}\right]^{-1}$$



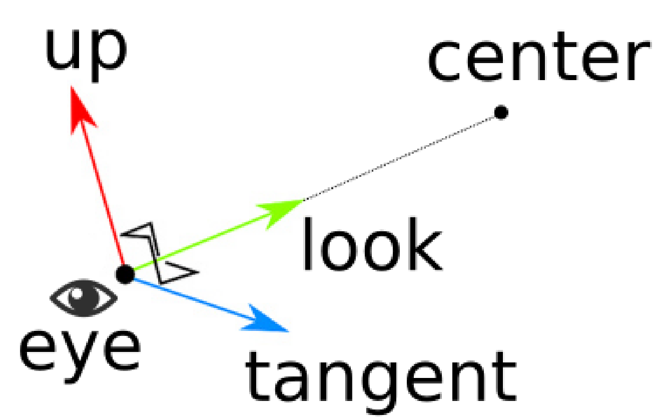
Building the View Matrix

Three axes: tangent, up, look

Note: camera looks down **negative** look direction for extra confusion

big source of bugs!

$$V = \left[\begin{array}{ccc|c} \text{tangent} & \text{up} & \text{-look} & \text{eye} \\ \hline 0 & 0 & 0 & 1 \end{array}\right]^{-1}$$

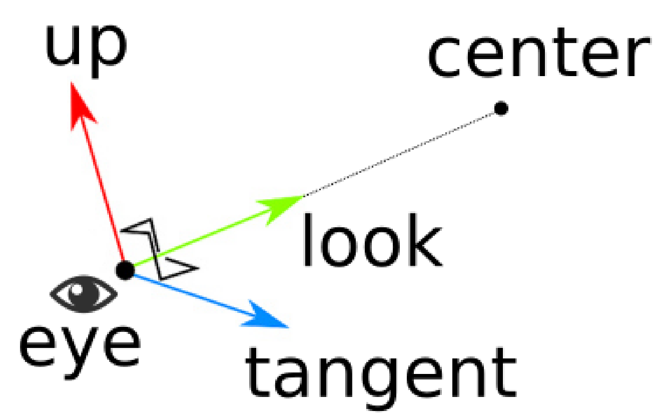


Building the View Matrix

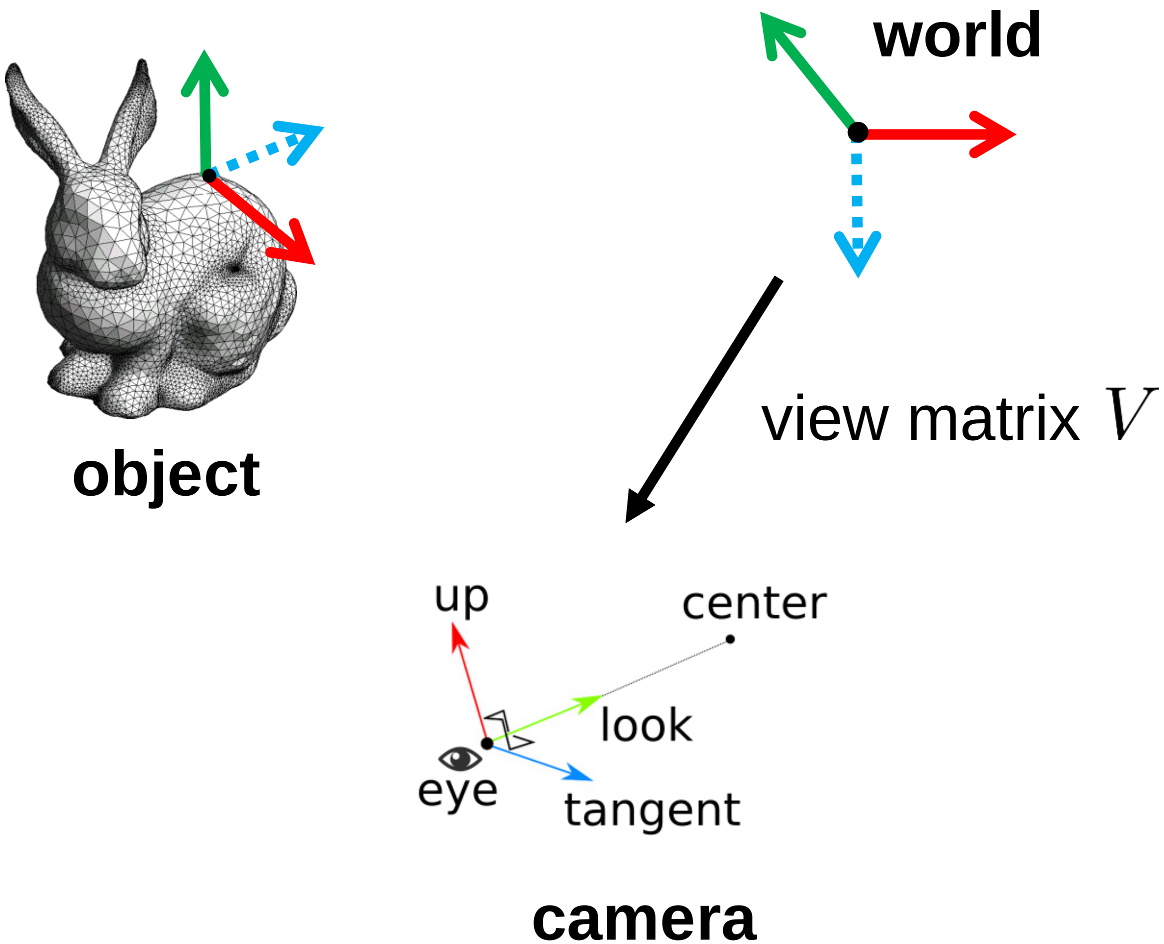
$$V = \left[\begin{array}{ccc|c} \multicolumn{3}{c|}{\overbrace{\begin{matrix}\text{tangent}\\ \text{up}\\ \text{-look}\end{matrix}}^{\text{R}}} & \text{-R eye} \\ \hline 0 & 0 & 0 & 1 \end{array}\right]$$

$$V = \left[\begin{array}{ccc|c} \text{tangent} & \text{up} & \text{-look} & \text{eye} \\ \hline 0 & 0 & 0 & 1 \end{array}\right]^{-1}$$

big source of bugs!



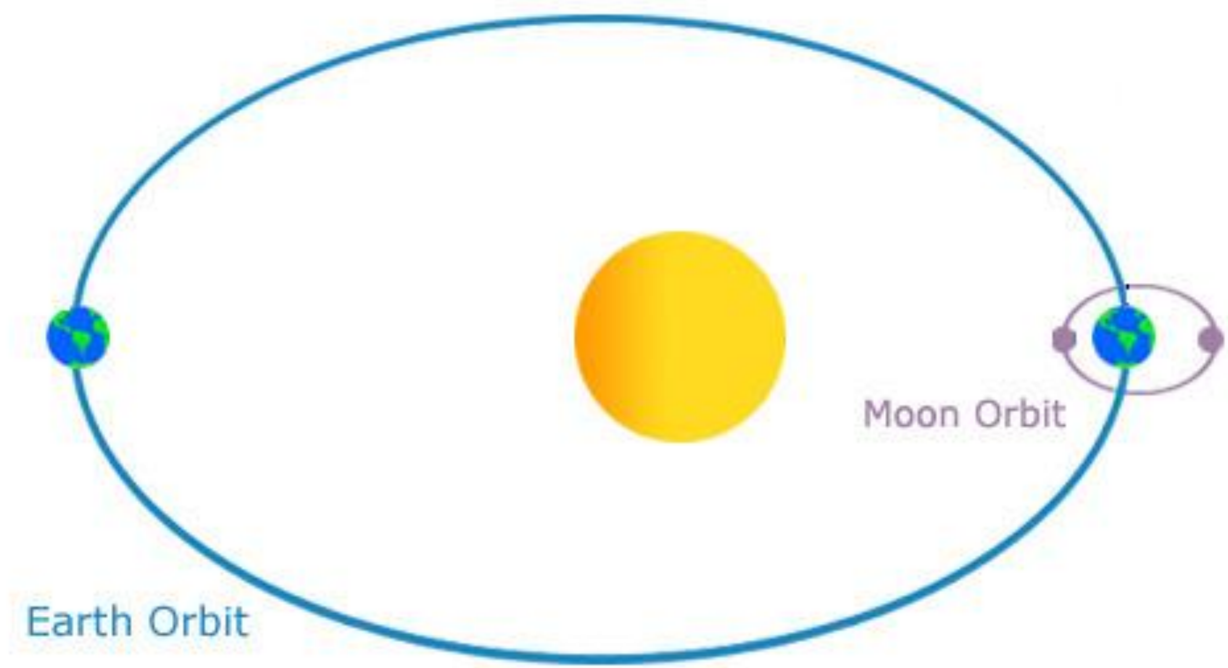
Coordinate Systems in Graphics



Why Use Object Coordinates?

Why Use Object Coordinates?

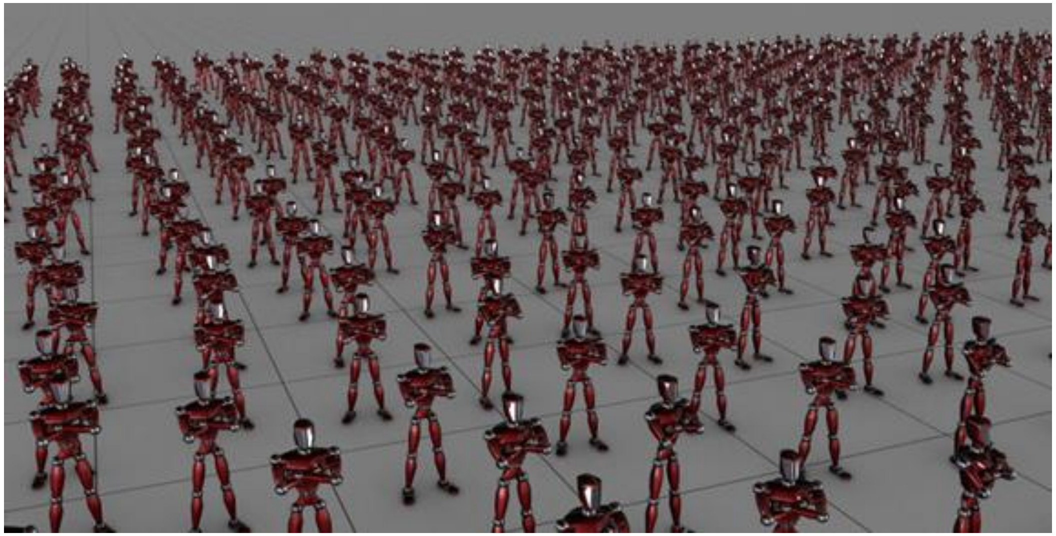
Easier to work with / animate



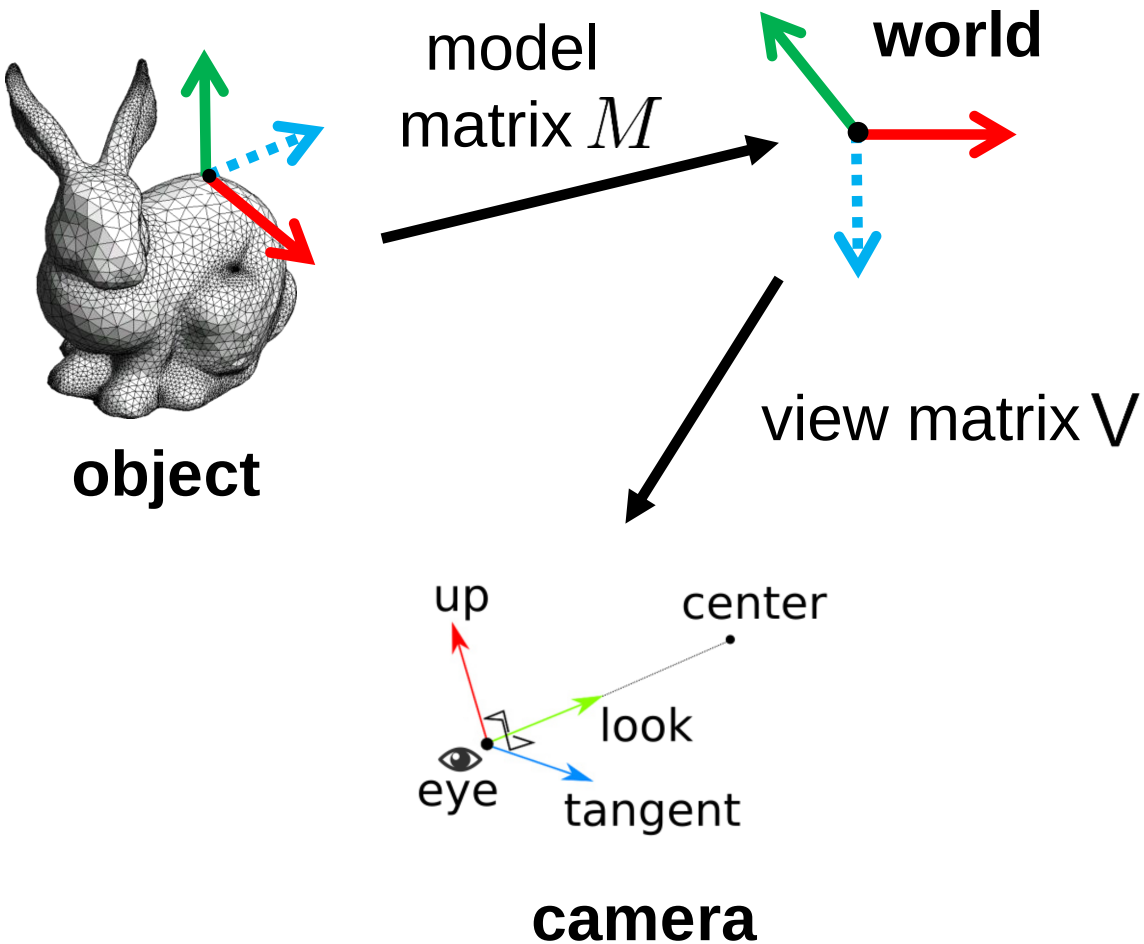
Why Use Object Coordinates?

Easier to work with / animate

Instancing

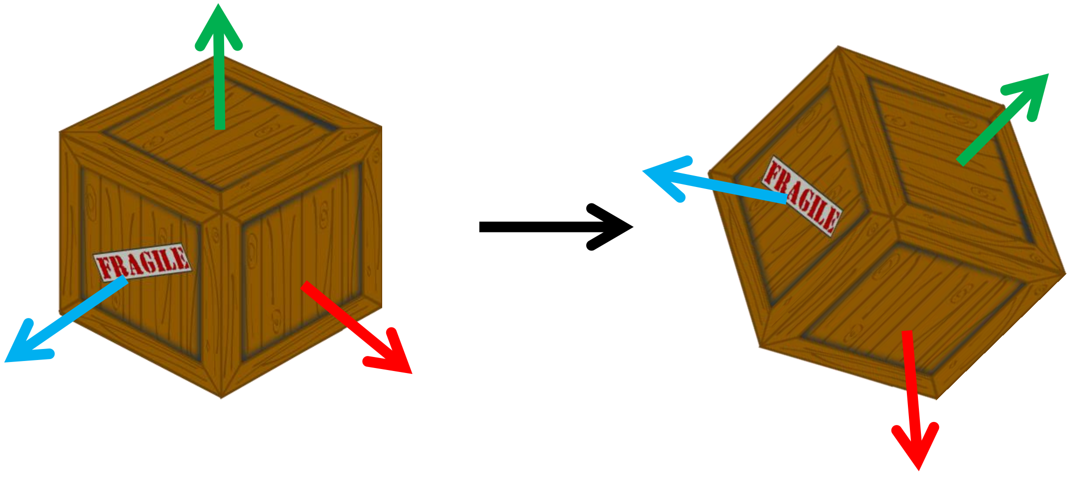


Coordinate Systems in Graphics



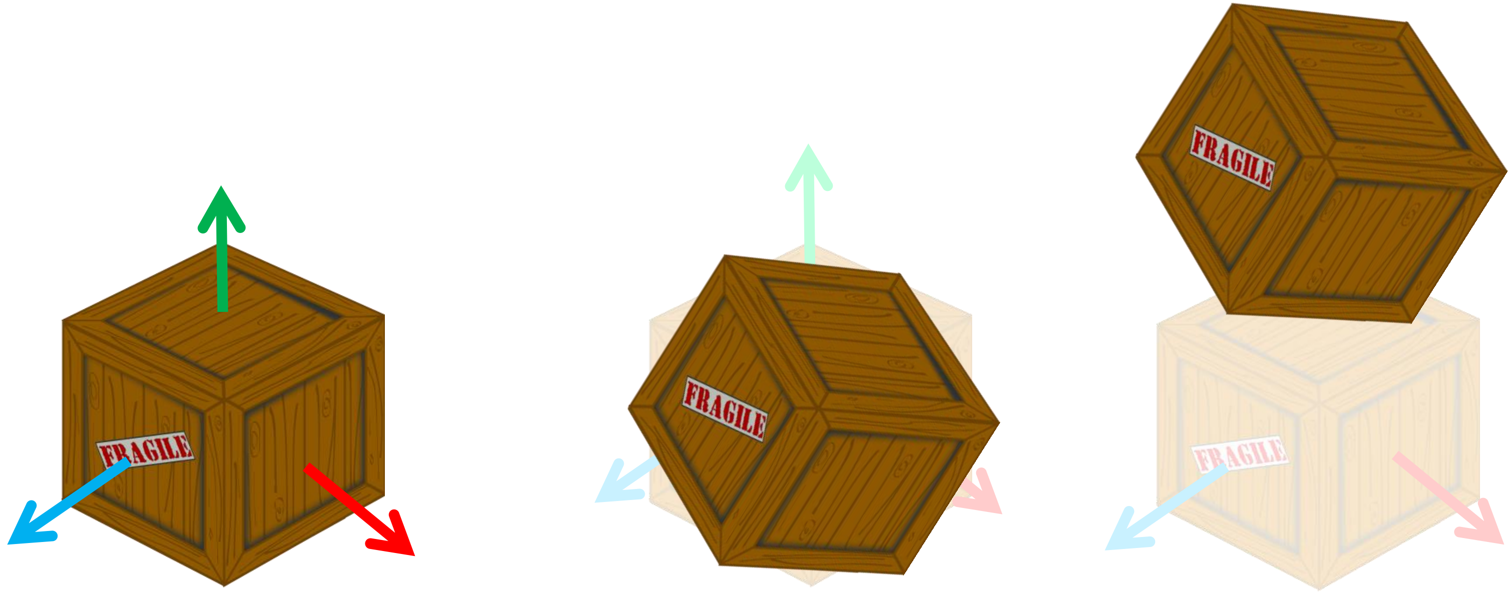
Transformations

Every transformation **creates child coordinate system**



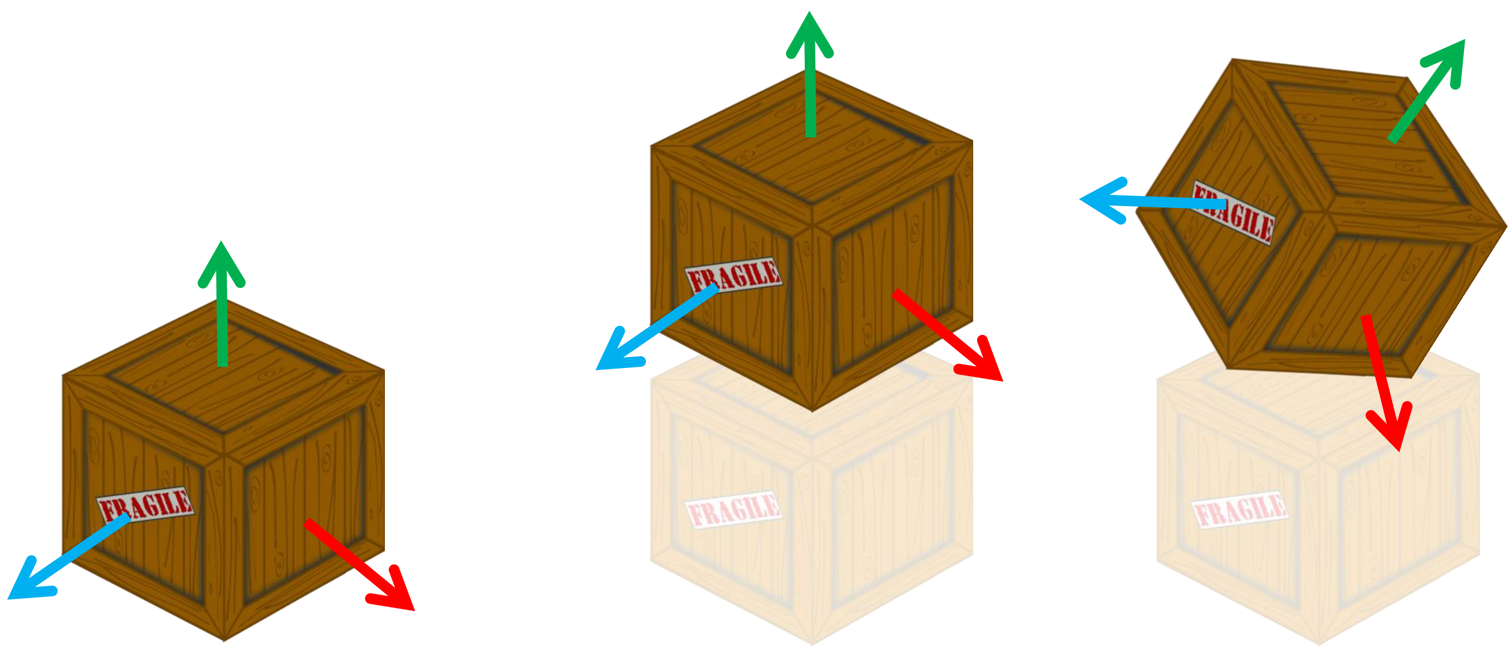
Two Interpretations of TR

Backwards: transforms applied **right to left** in original coordinate system



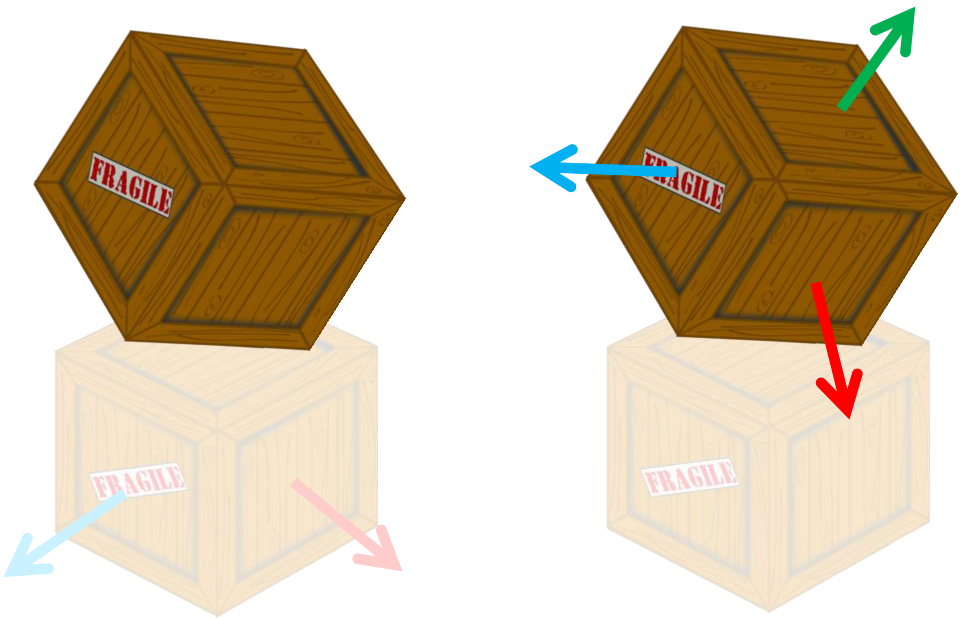
Two Interpretations of TR

Forwards: transforms applied **left to right** in new coordinate systems



Two Interpretations of TR

Same answer either way, but both interpretations useful



Scene Graph

Represents hierarchy of transformations

Assignment 3: bones in character body